

第 15 章 多媒体程序设计

多媒体是融合两种或者两种以上媒体的一种人机交互式信息交流和传播媒体，使用的媒体包括文字、图形、图像、声音、动画和电视图像（video）。本章将介绍如何使用 Java 提供的工具包来编写播放图像、音频、视频的程序。

15.1 声音文件的播放

声音是携带信息的极其重要的媒体，是多媒体技术研究中的一个重要内容。声音的种类繁多，如人的话音、乐器声、动物发出的声音、机器产生的声音以及自然界的雷声、风声、雨声、闪电声等。这些声音有许多共同的特性，也有它们各自的特性。在用计算机处理这些声音时，既要考虑它们的共性，又要利用它们各自的特性。

为了适应各种需要，声音的格式非常多，如 WAV、MP3、AU、AIFF、RMF、MIDI 等。作为 Java 应用程序员，并不需要掌握这些格式的解析，因为 Java 已经提供了现成的类来播放这些格式的文件。下面简要介绍一下各种声音文件格式的特点。

- ❑ AU（扩展名为 AU 或 SND）：适用于短的声音文件，为 Solaris 和下一代机器的通用文件格式，也是 Java 平台标准的音频格式。AU 类型文件使用的三种典型音频格式为：8 位 μ -law 类型（通常采样频率为 8kHz）、8 位线性类型，以及 16 位线性类型。
- ❑ WAV（扩展名为 WAV）：由 Microsoft 和 IBM 共同开发，对 WAV 的支持已经被加进 Windows 95 并且被延伸到后继的所有 Windows 操作系统。WAV 文件能存储各种格式，包括 μ -law、a-law 和 PCM（线性）数据。它们几乎能被所有支持声音的 Windows 应用程序播放。
- ❑ AIFF（扩展名为 AIF 或 IEF）：音频互换文件格式，是为 Macintosh 计算机和 Silicon Graphics（SGI）计算机所共用的标准音频文件格式。AIFF 和 AIFF-C 几乎是相同的，除了后者支持例如 μ -law 和 IMA ADPCM 类型的压缩。
- ❑ MIDI（扩展名为 MID）：乐器数字接口，MIDI 是为音乐制造业所认可的标准，主要用于控制诸如合成器和声卡之类的设备。MIDI 文件不包含数字音频采样，而是包括一系列指令，通过这些指令把来自不同乐器上的音符序列合成乐曲。一些 MIDI 文件包含附加指令来为各种合成设置进行编程。大多数合成器支持 MIDI 标准，所以在一个合成器上制作的音乐能够在另一个合成器上播放。有 MIDI 接口的计算机能处理 MIDI 数据以产生新音乐或音响效果。例如，一个完整的音乐作品可以通过一个软件驱动的命令转换成全新的形式。Java 声音引擎支持两种 MIDI 文件

类型：MIDI 类型 0 文件，包含仅仅一个序列，所有相关的乐器部分被包含在同一个逻辑“磁道”上。MIDI 类型 1 文件，包含多重的“磁道”使得不同的乐器被逻辑地分开，从而使对声音的操作和重组更加容易。

- ❑ RMF（扩展名为 RMF）：混合音乐格式，是由 Beatnik 设计出来的混合文件类型，通过交互式设定将 MIDI 和音频采样封装在一起。RMF 好比是一个所有音乐相关文件的容器。RMF 也包含对有关版权的详细文件说明的支持。RMF 文件可以包含多个由不同艺术家创作的，存储为，MIDI 类型或音频采样类型的作品，每个都关联着相关的版权信息。

Java 的标准类库中有两种方法可用于播放声音，一个是 AudioClip 接口，它在 java.applet 包中；一个是 AudioStream 和 AudioPlayer 配合使用，它们在 sun.audio 包中。前者只能用在 applet 中，后者可用在应用程序中。

15.1.1 在 Applet 中使用 AudioClip 播放声音

在 Applet 中，可以使用 AudioClip 来播放声音，它非常简单，只有三个方法：play()、loop()、stop()。所以编程也非常简单，但是功能也比较少。也许 Java 的设计者认为在网页中播放背景音乐，有这几个简单的功能就够用了。

由于 AudioClip 是接口，所以不能直接使用它来生成对象。但可以声明一个 AudioClip 变量，然后用 applet 类中的 getAudioClip() 来获取一个实例对象，再使用它的 play() 方法来播放声音。下面是个简单的例子。

【例 15.1】 利用 AudioClip 播放声音文件。

//-----文件名 playMusic.java，程序编号 15.1-----

```
import java.awt.event.*;
import javax.swing.*;
import java.applet.*;

public class playMusic extends Applet implements ActionListener{
    AudioClip clip=null;
    JButton playBtn,loopBtn,stopBtn;
    public void init(){
        playBtn=new JButton("播放");
        loopBtn=new JButton("循环");
        stopBtn=new JButton("停止");
        playBtn.addActionListener(this);
        loopBtn.addActionListener(this);
        stopBtn.addActionListener(this);
        add(playBtn);
        add(loopBtn);
        add(stopBtn);
        //获取一个对象实例，test.wav 是要播放的声音文件
        clip=getAudioClip(getCodeBase(),"test.wav");
    }
    public void actionPerformed(ActionEvent e){
        if (e.getSource()==playBtn)
```

```

        clip.play(); //播放声音
    else if(e.getSource()==loopBtn)
        clip.loop(); //循环播放
    else
        clip.stop(); //停止播放
    }
}

```

15.1.2 在 Application 中播放声音

如果要在 Application 中播放声音，就不能使用 AudioClip，而应该用 AudioStream 和 AudioPlayer 配合起来播放。它们的功能比 AudioClip 稍强一些，编程也稍微复杂一点。

AudioStream 和 AudioPlayer 不是 Java 标准包中的类，而是 sun.audio 包中的类。其中，AudioStream 需要定义对象才能使用，而 AudioPlayer 中有一个静态的 player 变量，它其中都是静态方法，可以直接使用（类似于 System.out 变量）。

它们的一般用法是，先用 AudioStream 创建一个音频流对象，而后将此对象作为参数传递给 AudioPlayer.player.start() 方法以便播放。虽然 AudioPlayer.player 中只有 start() 和 stop() 两个方法，但是 start() 方法会从音频流对象上次停止播放的位置开始播放，而不是从头开始，所以用 stop() 暂停一个音频流的播放后，可以使用 start() 继续播放。下面的例子演示了如何使用这两个方法实现播放、暂停、继续播放和停止 4 个功能。

【例 15.2】 在 Application 中播放声音文件。

//-----文件名 playAudio.java，程序编号 15.2-----

```

import javax.swing.*;
import java.awt.event.*;
import sun.audio.*; //AudioStream 和 AudioPlayer 在此包中
import java.awt.*;
import java.io.*;

public class playAudio implements ActionListener{
    protected JTextField fileField;
    protected JButton openBtn,startBtn,pauseBtn,resumBtn,stopBtn;
    protected Container con;
    protected JFrame mainJframe;
    protected AudioStream as; //声明音频流对象
    protected FileInputStream fileau;

    public playAudio(){
        mainJframe=new JFrame("播放声音");
        con=mainJframe.getContentPane();
        con.setLayout(new FlowLayout());
        fileField=new JTextField();
        fileField.setColumns(30);
        openBtn=new JButton("选择文件");
        startBtn=new JButton("开始播放");
        pauseBtn=new JButton("暂停播放");
    }
}

```

```

    resumBtn=new JButton("继续播放");
    stopBtn=new JButton("停止播放");
    openBtn.addActionListener(this);
    startBtn.addActionListener(this);
    pauseBtn.addActionListener(this);
    resumBtn.addActionListener(this);
    stopBtn.addActionListener(this);
    con.add(fileField);
    con.add(openBtn);
    con.add(startBtn);
    con.add(pauseBtn);
    con.add(resumBtn);
    con.add(stopBtn);
    mainJframe.setSize(400,200);
    mainJframe.setVisible(true);
    mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void actionPerformed(ActionEvent e) {
    Object obj;
    obj=e.getSource();
    try{
        if(obj==openBtn){                //让用户选择音频文件
            openfile();
        }else if(obj==startBtn){          //播放音频文件
            if (fileau!=null) fileau.close();
            fileau = new FileInputStream(fileField.getText());
            as=new AudioStream(fileau);    //创建音频流对象
            AudioPlayer.player.start(as); //从头开始播放音频流
        }else if(obj==pauseBtn){          //暂停音频流的播放
            AudioPlayer.player.stop(as);
        }else if(obj==resumBtn){           //继续从上次暂停的地方开始播放
            AudioPlayer.player.start(as);
        }else if(obj==stopBtn){           //停止音频流的播放
            AudioPlayer.player.stop(as);
            as.close();                    //关闭音频流
            fileau.close();
            fileau=null;
        }
    }catch(Exception el){
        JOptionPane.showMessageDialog(mainJframe,"无法播放文件!");
    }
}

private void openfile(){
    try{
        JFileChooser chooser = new JFileChooser();
        if(chooser.showOpenDialog(mainJframe)==JFileChooser.APPROVE_
            OPTION)
            fileField.setText(chooser.getSelectedFile().toString());
    }catch(Exception e){

```

```

        JOptionPane.showMessageDialog(mainJframe, "无法加载文件! ");
    }
}

public static void main(String args[]){
    new playAudio();
}
}

```

图 15.1 是程序运行时的截图。



图 15.1 程序运行时截图

15.1.3 利用 JavaSound API 播放声音

由于 AudioPlayer 的功能不是很强大，能支持的声音格式有限——它不支持最常见的音乐文件，如 MP3，所以从 JDK1.4 以后，Sun 公司又提供了一个 javax.sound.sampled 包，被称为 JavaSound API，该包的功能更强大，如果配合 JavaZoom 提供的兼容 JavaSound 的纯 Java 解码器——JavaLayer，就可以播放 MP3 格式的音乐。

该包中的类虽然灵活，但更为低级，编程更为困难，多数功能需要程序员自己来封装。而且为了在 GUI 中播放音乐时能让用户灵活地进行控制，播放音乐的方法必须封装在线程中，而要很好地控制线程，本身就对程序员是一个挑战。下面请看一个例子。

【例 15.3】 使用 Java Sound API 播放声音文件。

首先需要将这些 API 封装在一个线程中，下面的类完成了这一工作。

//-----文件名 SoundBase.java，程序编号 15.3-----

```

import java.io.File;
import java.io.IOException;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.DataLine;
import javax.swing.*;

//本类必须是一个线程类
public class SoundBase implements Runnable {
    private static final int BUFFER_SIZE = 1024;
    private String fileToPlay = null;
    //下面定义线程中通讯用的变量
    private static boolean threadExit = false;

```

```
private static boolean stopped = true;
private static boolean paused = false;
private static boolean playing = false;
//用于线程的同步管理
public static Object synch = new Object();
private Thread playerThread = null;

public SoundBase(String filename) {
    fileToPlay = filename;
}

public SoundBase() {
    fileToPlay = "default.wav";
}

public void run() {
    while (! threadExit) {
        waitForSignal();
        if (! stopped)
            playMusic();
    }
}

public void endThread() {
    threadExit = true;
    synchronized(synch) {
        synch.notifyAll();
    }
    try {
        Thread.sleep(500);
    } catch (Exception ex) {}
}

public void waitForSignal() {
    try {
        synchronized(synch){
            synch.wait();
        }
    }catch (Exception ex) { }
}

public void play() {
    if ((!stopped) || (paused)) return;
    if (playerThread == null) {
        playerThread = new Thread(this);
        playerThread.start();
        try {
            Thread.sleep(500);
        } catch (Exception ex) {}
    }
    synchronized(synch) {
```

```
        stopped = false;
        paused = false;
        synch.notifyAll();
    }
}

public void setFileToPlay(String fname) {
    fileToPlay = fname;
}

public void playFile(String fname) {
    setFileToPlay(fname);
    play();
}

public void playMusic() {
    byte[] audioData = new byte[BUFFER_SIZE];
    AudioInputStream ais = null;
    SourceDataLine line = null;
    AudioFormat baseFormat = null;
    try {
        ais = AudioSystem.getAudioInputStream(new File (fileToPlay));
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "打开文件失败!");
    }
    if (ais != null) {
        baseFormat = ais.getFormat(); //获取文件格式
        line = getLine(baseFormat);   //从文件中获取数据
        if (line == null) {           //如果不是可解码类型, 测试能否获取外部解
                                     码器
            AudioFormat decodedFormat = new AudioFormat(
                AudioFormat.Encoding.PCM_SIGNED,
                baseFormat.getSampleRate(),
                16,
                baseFormat.getChannels(),
                baseFormat.getChannels() * 2,
                baseFormat.getSampleRate(),
                false );
            ais = AudioSystem.getAudioInputStream(decodedFormat, ais);
            line = getLine(decodedFormat);
        }
    }
    if (line == null) return;         //不能播放此文件
    playing = true;
    line.start();                     //准备播放文件
    int inBytes = 0;
    //循环播放声音
    while ((inBytes != -1) && (!stopped) && (!threadExit)) {
        try {
            inBytes = ais.read(audioData, 0, BUFFER_SIZE);
        } catch (IOException e) {
```

```
        JOptionPane.showMessageDialog(null, "无法读取文件中的内容!");
    }
    try{
        if (inBytes > 0)
            line.write(audioData, 0, inBytes);
    }catch(Exception e) {
        JOptionPane.showMessageDialog(null, "无法输出解码数据到音频设备!");
    }
    if (paused)
        waitForSignal();
}
line.drain();
line.stop();
line.close();
playing = false;
stopped = true;
paused = false;
}

public void stop() {
    paused = false;
    stopped = true;
    waitForPlayToStop();
}

public void waitForPlayToStop() {
    while( playing)
        try {
            Thread.sleep(500);
            synchronized(synch) {
                synch.notifyAll();
            }
        } catch (Exception ex) { }
}

public void pause() {
    if (stopped) return;
    synchronized(synch) {
        paused = true;
        synch.notifyAll();
    }
}

public void resume(){
    if (stopped) return;
    synchronized(synch) {
        paused = false;
        synch.notifyAll();
    }
}
```



```

private SourceDataLine getLine(AudioFormat audioFormat) {
    SourceDataLine res = null;
    DataLine.Info info = new DataLine.Info(SourceDataLine.class,
        audioFormat);
    try {
        res = (SourceDataLine) AudioSystem.getLine(info);
        res.open(audioFormat);
    } catch (Exception e) {
        res = null;
    }
    return res;
}
}

```

在这个线程类中，用到的类非常多，除了用于线程控制的部分变量和对象，用于音频播放的类主要有两个：AudioInputStream 和 SourceDataLine。前者作为音频数据的输入流，后者将音频数据输出到 AudioFormat 中，由它来播放声音。

有了这个类做基础，编写 GUI 界面就比较简单了，程序如下：

//-----文件名 playMP3.java，程序编号 15.4-----

```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;

public class playMP3 implements ActionListener{
    protected JTextField fileField;
    protected JButton openBtn,startBtn,pauseBtn,resumBtn,stopBtn;
    protected Container con;
    protected JFrame mainJframe;
    protected SoundBase as;
    protected String filename;

    public playMP3(){
        mainJframe=new JFrame("播放声音");
        con=mainJframe.getContentPane();
        con.setLayout(new FlowLayout());
        fileField=new JTextField();
        fileField.setColumns(30);
        openBtn=new JButton("选择文件");
        startBtn=new JButton("开始播放");
        pauseBtn=new JButton("暂停播放");
        resumBtn=new JButton("继续播放");
        stopBtn=new JButton("停止播放");
        openBtn.addActionListener(this);
        startBtn.addActionListener(this);
        pauseBtn.addActionListener(this);
        resumBtn.addActionListener(this);
        stopBtn.addActionListener(this);
        con.add(fileField);
    }
}

```

```

        con.add(openBtn);
        con.add(startBtn);
        con.add(pauseBtn);
        con.add(resumBtn);
        con.add(stopBtn);
        mainJframe.setSize(400,200);
        mainJframe.setVisible(true);
        mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        as = new SoundBase();
    }

    public void actionPerformed(ActionEvent e) {
        Object obj;
        obj=e.getSource();
        try{
            if(obj==openBtn){
                openfile();
            }else if(obj==startBtn){
                as.playFile(filename);
            }else if(obj==pauseBtn){
                as.pause();
            }else if(obj==resumBtn){
                as.resume();
            }else if(obj==stopBtn){
                as.stop();
            }
        }catch(Exception el){
            JOptionPane.showMessageDialog(mainJframe,"无法播放文件! ");
        }
    }

    private void openfile(){
        try{
            JFileChooser chooser = new JFileChooser();
            if(chooser.showOpenDialog(mainJframe)==JFileChooser.APPROVE_
            OPTION){
                File tempfile= chooser.getSelectedFile();
                filename = tempfile.toString();
                fileField.setText(filename) ;
            }
        }catch(Exception el){
            JOptionPane.showMessageDialog(mainJframe,"无法加载文件! ");
        }
    }

    public static void main(String args[]){
        new playMP3();
    }
}

```

程序 15.4 的界面和前面程序 15.2 是完全一样的，但它可以播放 MP3 等格式的音乐。

不过，需要系统正确安装 MP3 等格式的解码器才能正常工作。

15.2 基本图形处理

在计算机学中，有一门专门的课程叫做图形学，专门介绍如何处理图形。即使是画出一些很简单的图形也需要一定的数学（主要是线性代数和解析几何）基础。Java 将这些基本图形以及处理算法都封装起来，方便应用程序员编程。这一节就来介绍如何处理一些基本的图形，包括：直线、矩形、圆和椭圆、多边形，以及如何缩放图形和填充封闭图形等。

在 14.7.1 小节中，曾经介绍如何将 JPanel 当作画布使用，本节的大多数程序仍然是在 JPanel 上面绘出图形。在 JPanel 上画图形的一般步骤是：

- (1) 获取 JPanel 的画笔；
- (2) 设置画笔的色彩等属性；
- (3) 在指定位置画出所需图形。

这里尽管是以 JPanel 为例，实际上如果要在 Applet 中画图，基本步骤也是一样的，只是获取画笔的方式更为简单一些。

Java 中的画笔是以类 Graphics 来表示，它是一个抽象类，封装了绝大多数基本图形的绘制方法。它有两个子类：DebugGraphics 和 Graphics2D。不过在多数情况下，并不需要直接使用这两个类，而只需声明一个 Graphics 的变量，而后获取由 JPanel 返回的画笔对象即可。

15.2.1 画直线

直线是最为基本的图形，大多数的其他图形都是由直线合成的。也许读者会感到疑惑：点不是比直线更为简单和基本吗？但实际情况是，如果用点来绘制复杂图形，速度要比用直线合成慢得多。在 Graphics 中，没有提供专门的画点的方法，而只能用画圆形来实现画点，这样一来，速度就更慢了。

Graphics 中画直线的方法声明如下：

```
abstract void drawLine(int x1, int y1, int x2, int y2)
```

它会用当前画笔的色彩画一根从 (x1, y1) 到 (x2, y2) 的直线。其中的 (x1, y1) 和 (x2, y2) 都是按照图 15.2 所示的控件坐标系中的坐标：

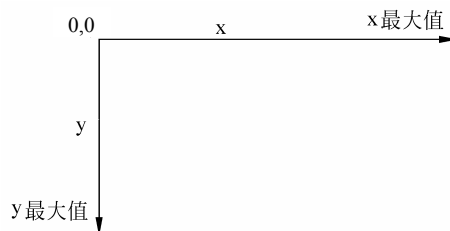


图 15.2 控件坐标系

和一般的笛卡尔坐标系相比，它的 y 值增长方向是相反的。另外一个不同是，x 和 y 值都必须是整数，它们的最大值受限于屏幕的分辨率。例如，屏幕分辨率为 1024×768，则 x 最大值为 1023，y 的最大值为 767。

在画图之前还要设置画笔的色彩，这需要用下面的方法：

```
abstract void setColor(Color c)
```

其中，c 是一个 Color 对象，用它来指定画笔的色彩。设置色彩后，对其后所有的图形都有效，但对前面已经绘制的图形色彩没有影响。如果要改变色彩，可以重新调用本方法。

创建一个 Color 对象的构造方法很多，其中最常用的是下面的构造方法：

```
Color(int r, int g, int b)
```

其中，r、g、b 三个整数分别指示了色彩的三个分量：红、绿、蓝，三个数值都是从 0～255。如果想使用某些常用的色彩，也可以使用 Color 类定义好的一些静态常量值，例如，red、green、blue 等。它们都是 Color 类型，这样就无需创建 Color 对象而可以直接指定图形的色彩。

【例 15.4】 画直线示例。

//-----文件名 DrawLines.java，程序编号 15.5-----

```
import java.awt.*;
import javax.swing.*;
public class DrawLines{
    MyCanvas palette;
    JFrame mainJFrame;
    public class MyCanvas extends JPanel{           //继承并扩展 JPanel 类
        //当控件发生变化时，系统会调用此方法，图形可以在此方法中绘制
        public void paintComponent(Graphics g){     //g 是系统传递进来的画笔
            g.setColor(Color.red);                  //设置画笔的色彩为红色
            g.drawLine(0,0,300,300);                //从左上角到右下角画直线
            g.setColor(Color.blue);                  //设置画笔的色彩为蓝色
            g.drawLine(0,300,300,0);                //从左下角到右上角画直线
            g.setColor(Color.green);                 //设置画笔的色彩为绿色
            g.drawLine(150,0,150,300);              //从上到下画垂直直线
            g.setColor(Color.yellow);                //设置画笔的色彩为黄色
            g.drawLine(0,150,300,150);              //从左到右画水平直线
        }
    }
    public DrawLines(){
        mainJFrame=new JFrame("画直线示例");
        palette=new MyCanvas();
        mainJFrame.getContentPane().add(palette);
        mainJFrame.setSize(310,310);
        mainJFrame.setVisible(true);
        mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){
        new DrawLines();
    }
}
```

程序运行截图如图 15.3 所示。

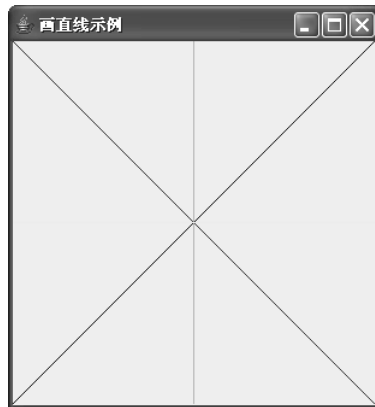


图 15.3 程序运行截图

15.2.2 画矩形

Graphics 中提供了画矩形的方法，它以画笔的当前颜色画出矩形，原型声明如下：

```
void drawRect(int x, int y, int width, int height)
```

其中，x 和 y 表示矩形左上角的坐标，width 和 height 分别表示宽度和高度。当二者相等时，画出来的是正方形。

如果要画出圆角矩形，需要用 drawRoundRect() 方法，原型声明如下：

```
abstract void drawRoundRect(int x, int y, int width, int height,  
                             int arcWidth, int arcHeight)
```

前面 4 个参数的意义与 drawRect() 相同，arcWidth 表示 4 个圆角水平方向的直径长度，arcHeight 表示它们垂直方向的直径长度。如果这两个值相等，那么这是一段圆弧，否则是椭圆弧。

【例 15.5】 画矩形示例。

//-----文件名 DrawRects.java, 程序编号 15.6-----

```
import java.awt.*;  
import javax.swing.*;  
public class DrawRects{  
    MyCanvas palette;  
    JFrame mainJFrame;  
    public class MyCanvas extends JPanel{  
        public void paintComponent(Graphics g){  
            g.setColor(Color.red);           //设置画笔的色彩为红色  
            g.drawRect(10,10,200,100); //画一个长为 200，宽为 100 的长方形  
            g.setColor(Color.blue);          //设置画笔的色彩为蓝色  
            g.drawRoundRect(10,130,200,100,10,10); //画一个长为 200，宽为 100 的圆  
                                                    角长方形  
        }  
    }  
}
```

```

}
public DrawRects(){
    mainJFrame=new JFrame("画矩形示例");
    palette=new MyCanvas();
    mainJFrame.getContentPane().add(palette);
    mainJFrame.setSize(310,310);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args){
    new DrawRects();
}
}

```

程序运行截图如图 15.4 所示。

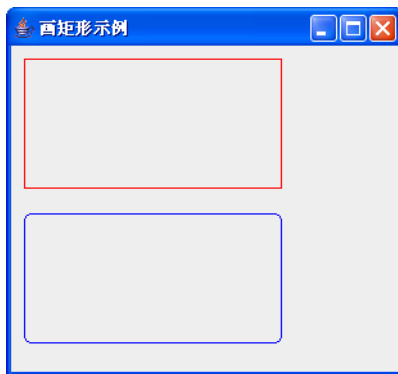


图 15.4 程序运行截图

15.2.3 画椭圆和圆弧

在 Graphics 中提供了一个方法来画椭圆和圆形，它的原型声明如下：

```
abstract void drawOval(int x, int y, int width, int height)
```

其中，x 和 y 是椭圆外切矩形的左上角的坐标，width 是它的宽度，height 是它的高度。当宽度和高度相等时，画出来的就是圆形。

画圆弧的方法要复杂一些，它的声明如下：

```
abstract void drawArc(int x, int y, int width, int height,
                     int startAngle, int arcAngle)
```

它的前面 4 个参数与 drawOval() 相同。startAngle 表示圆弧起点的角度，0 表示在水平位置，intarcAngle 表示圆弧绕过的角度。如果为正值，表示按逆时针绘制圆弧；如果为负值，则按顺时针绘制圆弧。它是以角度为单位，如果该值为 360，则画出来的是一个完整的椭圆或者圆形。由此可见，drawOval() 是 drawArc() 的特殊情形。

【例 15.6】 画椭圆和圆弧示例。

//-----文件名 DrawArcs.java，程序编号 15.7-----

```

import java.awt.*;
import javax.swing.*;
public class DrawArcs{
    MyCanvas palette;
    JFrame mainJFrame;
    public class MyCanvas extends JPanel{
        public void paintComponent(Graphics g){
            g.setColor(Color.red);           //设置画笔的色彩为红色
            g.drawOval(10,10,140,70);         //画一个宽为 140，高为 70 的椭圆
            g.drawOval(160,10,100,100);       //画一个直径为 100 的圆
            g.drawArc(10,120,140,140,0,180); //画一个直径 140 的上半圆
            g.drawArc(160,120,140,80,180,180); //画一个宽 140、高 80 的下半椭圆弧
        }
    }
    public DrawArcs(){
        mainJFrame=new JFrame("画椭圆和圆弧示例");
        palette=new MyCanvas();
        mainJFrame.getContentPane().add(palette);
        mainJFrame.setSize(310,310);
        mainJFrame.setVisible(true);
        mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){
        new DrawArcs();
    }
}

```

程序运行截图如图 15.5 所示。



图 15.5 程序运行截图

15.2.4 画多边形

多边形是由若干段首尾相连的直线段组成的封闭图形。在 `Graphics` 中可以使用下面的方法来画多边形：

```

abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)

```

其中, 数组 `xPoints` 和 `yPoints` 分别表示线段端点的 `x` 和 `y` 坐标值, `nPoints` 表示端点的数目。任意一条线段由 `(xPoints[i-1],yPoints[i-1])` 和 `(xPoints[i],yPoints[i])` 来确定。如果给定的端点不是闭合的, 此方法将自动在最后一个端点后画出一条线段与第一个端点连接, 将整个图形闭合。

另外一个类似的方法是:

```
void drawPolygon(Polygon p)
```

所要绘出的多边形由参数 `p` 来决定, 参数 `p` 是 `Polygon` 类型的对象, 该类也存储了一系列的端点。

【例 15.7】 画多边形示例。

//-----文件名 DrawPoly.java, 程序编号 15.8-----

```
import java.awt.*;
import javax.swing.*;
public class DrawPoly{
    MyCanvas palette;
    JFrame mainJFrame;
    int xPoints[] = {30,200,30,200,30};
    int yPoints[] = {30,30,200,200,30};
    public class MyCanvas extends JPanel{
        public void paintComponent(Graphics g){
            g.setColor(Color.red); //设置画笔的色彩为红色
            g.drawPolygon(xPoints,yPoints,xPoints.length); //画出多边形
        }
    }
    public DrawPoly(){
        mainJFrame=new JFrame("画多边形示例");
        palette=new MyCanvas();
        mainJFrame.getContentPane().add(palette);
        mainJFrame.setSize(310,310);
        mainJFrame.setVisible(true);
        mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){
        new DrawPoly();
    }
}
```

程序运行截图如图 15.6 所示。

15.2.5 封闭图形的填充

前面介绍的图形都只绘出了轮廓, 有时候需要将其封闭的区域填充某种颜色。要完成这一功能, 可以使用 `Graphics` 提供的以 `fill` 开头的系列方法, 它们的原型声明如下:

❑ `void fillArc(int x, int y, int width, int height, int startAngle,`

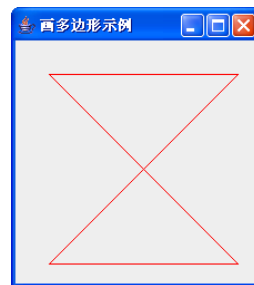


图 15.6 程序运行截图

int arcAngle): 绘出圆弧并填充。

- ❑ void fillOval (int x, int y, int width, int height): 绘出椭圆并填充。
- ❑ void fillPolygon (int[] xPoints, int[] yPoints, int nPoints): 绘出多边形并填充。
- ❑ void fillPolygon (Polygon p): 绘出多边形并填充。
- ❑ void fillRect (int x, int y, int width, int height): 绘出矩形并填充。
- ❑ void fillRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight): 绘出圆角矩形并填充。

每一个以 fill 开头的方法和对应的以 draw 开头的方法参数含义完全相同, 绘出的图形也相同, 只是画完图形后, 会用画笔当前的颜色将其填充。对于圆弧, 填充区域是起始点、结束点和圆心组成的封闭区域。

【例 15.8】 填充图形示例。

//-----文件名 FillArea.java, 程序编号 15.9-----

```
import java.awt.*;
import javax.swing.*;
public class FillArea{
    MyCanvas palette;
    JFrame mainJFrame;
    int xPoints[] = {30,100,30,100,30};
    int yPoints[] = {30,30,100,100,30};
    public class MyCanvas extends JPanel{
        public void paintComponent(Graphics g){
            g.setColor(Color.red);                //设置画笔的色彩为红色
            g.fillPolygon(xPoints,yPoints,xPoints.length);//画出多边形并填充
            g.fillArc(110,30,140,140,0,90);        //画一个直径 140 的 1/4 圆并填充
            g.fillRect(30,150,200,100);            //画一个长为 200, 宽为 100 的长方形并填充
            g.setColor(Color.blue);                //重新设置画笔色彩为蓝色
            g.drawRect(30,150,200,100);            //画一个长为 200, 宽为 100 的长方形轮廓
        }
    }
    public FillArea(){
        mainJFrame=new JFrame("图形填充示例");
        palette=new MyCanvas();
        mainJFrame.getContentPane().add(palette);
        mainJFrame.setSize(310,310);
        mainJFrame.setVisible(true);
        mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){
        new FillArea();
    }
}
```

程序运行截图如图 15.7 所示。

由于填充图形时所用的色彩和边缘色彩是一样的，如果希望边缘色彩与内部色彩不同，则需要分两次来绘制这个图形：先用某种色彩填充内部图形，再用另外一种色彩画出外部轮廓。在程序 15.9 中，最后画矩形就是用的这种方法。它的外部轮廓是蓝色，内部颜色是红色。

15.2.6 缩放图形

前面绘制的图形大小是固定的，无论窗口大小如何变换，图形的位置和大小都不会发生变化。如果窗口区域比图形小，则图形超过的部分不会画出来。

在下面的例子中，程序绘制的图形会随着窗口的大小而变化。要做到这一点并不难，当窗口的大小发生变化时，系统会调用 `paintComponent()` 方法，只要在此方法中获取窗口的大小，并按照一定的算法，重新计算图形应该绘出的大小再重新绘制就可以了。

【例 15.9】 缩放图形示例。

//-----文件名 ResizeOval.java, 程序编号 15.10-----

```
import java.awt.*;
import javax.swing.*;
public class ResizeOval{
    MyCanvas palette;
    JFrame mainJFrame;
    public class MyCanvas extends JPanel{
        public void paintComponent(Graphics g){
            int height, width;
            //获取窗口大小，并计算椭圆的大小
            height = getHeight();
            width = getWidth();
            g.setColor(Color.red);
            //按照窗口大小绘制并填充椭圆
            g.fillOval(0,0,width,height);
        }
    }
    public ResizeOval(){
        mainJFrame=new JFrame("画直线示例");
        palette=new MyCanvas();
        mainJFrame.getContentPane().add(palette);
        mainJFrame.setSize(310,310);
        mainJFrame.setVisible(true);
        mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){
```

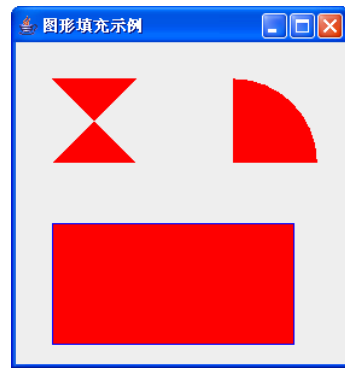


图 15.7 程序运行截图

```
new ResizeOval();  
}  
}
```

程序运行截图如图 15.8 所示。

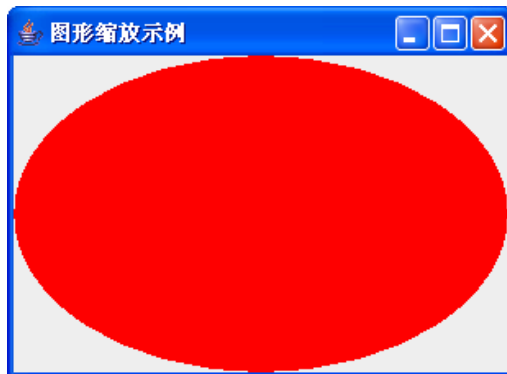


图 15.8 程序运行截图

无论用户如何调整这个窗口，椭圆都会恰好填充整个窗口区域。当然，这个例子非常简单，如果是多边形，那么计算它的大小就会复杂得多。

15.3 特效处理

15.2 节介绍的只是一些基本图形，但是如果将这些图形组合起来，并辅以某些色彩，将会达到一些令人意想不到的效果，人们将其称为特效。

15.3.1 颜色处理的基本知识

在 15.2 节中，每次画图之前都会用 `setColor()` 方法为画笔设置当前色彩，该方法的参数是一个 `Color` 类型的对象。为了简单起见，前面使用的都是该类预先定义好的一些常用色彩。实际上，该类提供了多个构造方法，并且对色彩提供了两种不同的合成模式。其中一种是前面介绍过的 RGB（红、绿、蓝三原色）模式，另外一种是 HSB（色调-饱和度-亮度）模式。RGB 模式使用比较多，也比较简单，但在实现某些特效时，使用 HSB 模式，编程更为简单一些。

在 HSB 模式中，色调是一个色彩环，取值在 0.0~1.0 之间，颜色有：赤、橙、黄、绿、青、蓝、紫。饱和度也在 0.0~1.0 之间，表示色调的浓和淡。亮度取值也在 0.0~1.0 之间，其中 1 是亮白色，0 是黑色。`Color` 中提供了两种将 HSB 与 RGB 互相转换的方法，它们原型声明如下：

- ❑ `static int HSBtoRGB (float hue, float saturation, float brightness)`：返回一个与参数（HSB）决定的色彩相同的 RGB 颜色值
- ❑ `static float[] RGBtoHSB (int r, int g, int b, float[] hsbvals)`：返回一个与参数（RGB）

决定的色彩相同的 HSB 颜色值，放在数组 `hsbvals` 中返回；如果 `hsbvals` 为 `null`，则新建一个数组返回这些值。

要用 HSB 模式创建一个 `Color` 对象，需要使用它的静态方法：

```
static Color getHSBColor(float h, float s, float b)
```

其中，参数 `h`、`s`、`b` 分别是色调、饱和度和亮度。

`Color` 中还定义了一些方法用于获得各种三原色的各个分量，比如：`getBlue()`、`getGreen()`、`getRed()`和 `getRGB()`。使用非常简单，不做更多的介绍。

与色彩相关的另外一个特性是色彩透明度。一般情况下，先画某种色彩后，在同一位置再画另外一种色彩，会将前面的色彩覆盖掉。我们将前面画出的色彩称为背景色，后面画出的色彩称为前景色。如果希望前景色不是完全覆盖背景色，而是在某种程度上和背景色混合起来，就需要用到色彩透明度。这一特性用数值 `alpha` 来描述，它是一个从 0.0 到 1.0 之间的浮点数，当它等于 0.0 时，表示完全透明；等于 1.0 时，则完全不透明。

`Graphics` 没有提供对透明度特性的支持，如果要使用该特性，需要使用 `Graphics` 的直接子类 `Graphics2D`。关于它的具体使用，将在 15.3.3 小节讲述。

15.3.2 淡入淡出效果

淡入淡出效果是最常见的特效之一。要实现淡入淡出效果方法非常简单，只要采用 HSB 模式设置色彩，其中的色调和亮度不必变化，只要调整饱和度的值即可。如果是淡入，则让饱和度逐步增加；如果是淡出，则让饱和度逐步减少。

【例 15.10】 淡入淡出效果示例。

//-----文件名 fadeInOut.java，程序编号 15.11-----

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class fadeInOut {
    JPanel palette;
    JFrame mainFrame;
    JButton btn;
    Container con;
    dealPushBtn handleBtn;
    //定义一个内部类来响应按钮事件
    public class dealPushBtn implements ActionListener{
        public void actionPerformed(ActionEvent e){
            Graphics g = palette.getGraphics(); //注意画笔的获取方式
            float h = 0.0f; //色彩为红色
            float s = 0.0f; //初始饱和度为 0，即没有颜色
            float b = 1.0f; //亮度为最大
            for(int i=0;i<100;i++){
                g.setColor(Color.getHSBColor(h,s,b)); //用 HSB 模式设置色彩
                g.fillRect(0,50,300,300); //显示色彩
            }
        }
    }
}
```

```

        try{
            Thread.sleep(50);
        }catch(InterruptedException e1){ }
        s += 0.01;    //增加饱和度, 让颜色逐步浓起来
    }
    for(int i=0;i<100;i++){
        g.setColor(Color.getHSBColor(h,s,b));
        g.fillRect(0,50,300,300);
        try{
            Thread.sleep(50);
        }catch(InterruptedException e1){ }
        s -= 0.01;    //减少饱和度, 让颜色逐步淡下去
    }
}
}

public fadeInOut(){
    mainFrame=new JFrame("色彩淡入淡出示例");
    handleBtn=new dealPushBtn();
    btn=new JButton("开始");
    btn.addActionListener(handleBtn);
    palette=new JPanel();
    palette.add(btn);
    mainFrame.getContentPane().add(palette);
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    palette.setLayout(new FlowLayout());
    mainFrame.setSize(300,300);
    mainFrame.setVisible(true);
}

public static void main(String[] args){
    new fadeInOut();
}
}

```

当单击“开始”按钮后, 色彩一开始是白色, 慢慢的变红, 直到变成大红色。然后再慢慢变淡, 直到变成白色。程序运行截图如图 15.9 所示。

这里色彩是采用的 HSB 模式, 如果采用 RGB 模式, 要实现同样的效果, 需要通过调整 alpha 值来实现, 编程要麻烦得多。

15.3.3 透明效果

本节来介绍如何设置前景色的透明度, 使得前景色和后景色能够混合起来。这需要使用 Graphics2D 类, 在其中声明了一个方法:



图 15.9 程序运行截图

```
abstract void setComposite(Composite comp)
```

其中, 参数 `comp` 是 `Composite` 类型的对象, 它可以用来设置 `alpha` 的值。但 `Composite` 本身是一个接口, 它的直接子类 `AlphaComposite` 有一个静态方法:

```
static AlphaComposite getInstance(int rule, float alpha)
```

可以创建一个具有指定 `alpha` 值的 `Composite` 类型的对象。

联合使用这些方法, 就可以设置所需要的透明度了。下面的例子演示了如何使用 `alpha` 值来实现色彩混合效果。

【例 15.11】 色彩混合效果示例。

//-----文件名 TransparencyExample.java, 程序编号 15.12-----

```
//定义自己的画布
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;

public class TransparencyExample extends JPanel {
    private static int gap=10, width=40, offset=10,
                    deltaX=gap+width+offset;
    private Rectangle
        blueSquare = new Rectangle(gap+offset, gap+offset, width, width),
        redSquare = new Rectangle(gap, gap, width, width);
    //创建一个指定 alpha 值的 AlphaComposite 对象
    private AlphaComposite makeComposite(float alpha) {
        int type = AlphaComposite.SRC_OVER;
        return(AlphaComposite.getInstance(type, alpha));
    }
    //用指定的 alpha 值来绘制前景色
    private void drawSquares(Graphics2D g2d, float alpha) {
        Composite originalComposite = g2d.getComposite();
        //用默认透明度绘制背景蓝色
        g2d.setPaint(Color.blue);
        g2d.fill(blueSquare);
        //设置透明度, 准备绘制前景红色
        g2d.setComposite(makeComposite(alpha));
        g2d.setPaint(Color.red);
        g2d.fill(redSquare);
        //将透明度设置回默认的模式
        g2d.setComposite(originalComposite);
    }
    //分别用不同的透明度来绘制颜色
    public void paintComponent(Graphics g) {
```

```

super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
for(int i=0; i<11; i++) {
    //alpha 值逐步增大, 透明度逐步减小
    drawSquares(g2d, i*0.1F);
    g2d.translate(deltaX, 0);
}
}
}

```

//-----文件名 mixing.java, 程序编号 15.13-----

```

import javax.swing.*;
import java.awt.*;
public class mixing{
    JFrame mainFrame;
    TransparencyExample palette;
    public mixing(){
        mainFrame=new JFrame("色彩混合示例");
        palette=new TransparencyExample();
        mainFrame.getContentPane().add(palette);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setSize(500,200);
        mainFrame.setVisible(true);
    }
    public static void main(String[] args) {
        new mixing();
    }
}

```

程序运行截图如图 15.10 所示。



图 15.10 程序运行截图

从图 15.10 中可以看出, 背景蓝色始终不变, 前景红色由淡转浓, 特别是与蓝色重合的部分, 逐渐由蓝色变为红色, 透明效果变化非常明显。

注意: 这里色彩的混合是通过透明来实现的, 它的效果与在 RGB 模式中混合红、蓝两种原色的效果并不相同。

15.3.4 盖房子特效

本节介绍用各种颜色的线形来画出一间房子。理论上来说,可以用直线画出任何图形,但是多数图形画起来过于复杂,所以多半不会用这种方法。但房子不同,它本身就是由一些直线组成,所以画起来相对而言要简单一些。

当然用画直线的方法来组成一幅图片,更多的不是需要编程能力,而是美工水平。

【例 15.12】 盖房子。

//-----文件名 houseCanvas.java, 程序编号 15.14-----

```
import javax.swing.*;
import java.awt.*;

public class houseCanvas extends JPanel{
    public houseCanvas(){
        setBackground(Color.black);
    }
    public void paintComponent(Graphics g){
        calc(g, 16, 110, 95, 0, 0, 0);
    }
    void calc(Graphics g, int i, int j, int k, int l, int i1, int j1){
        int k1 = i1;
        int l1 = l;
        int i2 = j1;
        int j2 = i;
        int k2 = l + i;
        int l2 = i1 + i;
        int i3 = j1 + i;
        int j3 = j2 << 1;
        int k3 = j3 << 1;
        for(int l3 = 0; l3 < 8; l3++){
            if(l3 == 1){
                j -= j3;
                k += j2;
                l1 = k2;
            }
            if(l3 == 2){
                j += k3;
                l1 = l;
                k1 = l2;
            }
            if(l3 == 3){
                j -= j3;
                k += j2;
                l1 = k2;
                k1 = l2;
            }
            if(l3 == 4){
                k -= k3;
            }
        }
    }
}
```



```
        l1 = 1;
        k1 = i1;
        i2 = i3;
    }
    if(l3 == 5){
        j -= j3;
        k += j2;
        l1 = k2;
        k1 = i1;
        i2 = i3;
    }
    if(l3 == 6){
        j += k3;
        l1 = 1;
        k1 = l2;
        i2 = i3;
    }
    if(l3 == 7){
        j -= j3;
        k += j2;
        l1 = k2;
        k1 = l2;
        i2 = i3;
    }
    if(i == 1)
        draw(l3, g, j, k, l1, k1, i2, j2);
    else
        calc(g, i >> 1, j, k, l1, k1, i2);
}

}

void draw(int i, Graphics g, int j, int k, int l, int i1, int j1, int
k1){
    boolean flag = false;
    byte byte0 = 1;
    byte byte1 = 4;
    int ai[] = new int[byte1];
    int ail[] = new int[byte1];
    byte byte2 = 63;
    byte byte3 = 127;
    char c = '\200';
    boolean flag1 = false;
    boolean flag2 = false;
    int l1 = k1 << 1;
    flag = false;
    byte0 = 1;
    if(l == 1)
        flag = true;
    if(l == 30)
        flag = true;
    if(i1 == 0)
```

```
        flag = true;
    if(i1 == 30)
        flag = true;
    if(j1 < 3)
        flag = true;
    if(i1 % 4 == 0 && j1 == 14)
        flag = true;
    if(i1 == 30 && j1 > 16 && l + j1 < 47)
        flag = true;
    if(i1 == 1 && j1 > 16 && l + j1 < 47)
        flag = true;
    if(i1 == 30 && j1 > 16 && j1 - l < 16)
        flag = true;
    if(l + j1 > 48)
        flag = false;
    if(j1 - l > 16)
        flag = false;
    if(l > 11 && l < 17 && i1 == 30 && j1 < 13 && j1 > 1)
        flag = false;
    if(l > 4 && l < 8 && i1 == 30 && j1 < 14 && j1 > 5)
        flag = false;
    if(l > 20 && l < 25 && i1 == 30 && j1 < 14 && j1 > 5)
        flag = false;
    if(l + j1 == 48){
        flag = true;
        byte0 = 2;
    }
    if(j1 - l == 16){
        flag = true;
        byte0 = 2;
    }
    if(i1 < 4 && l > 27 && j1 < 27)
        flag = true;
    if(i1 > 0 && i1 < 3 && l > 28 && l < 31 && j1 < 30)
        flag = true;
    if(flag){
        char c1;
        byte byte4;
        byte byte5;
        if(byte0 == 2){
            c1 = '\0';
            byte4 = 38;
            byte5 = 25;
        } else{
            c1 = '\200';
            byte4 = 5;
            byte5 = 25;
        }
        ai[0] = j;
        ai1[0] = k - 11;
        ai[1] = j - 11;
```

```

        ail[1] = k - 3 * k1;
        ai[2] = j;
        ail[2] = k - 4 * k1;
        ai[3] = j + 11;
        ail[3] = k - 3 * k1;
        g.setColor(new Color(byte3 + c1, byte3 + byte5, byte3 + byte4));
        g.fillPolygon(ai, ail, byte1);
        ai[0] = j;
        ail[0] = k;
        ai[1] = j - 11;
        ail[1] = k - k1;
        ai[2] = j - 11;
        ail[2] = k - 3 * k1;
        ai[3] = j;
        ail[3] = k - 11;
        g.setColor(new Color(byte2 + c1, byte2 + byte5, byte2 + byte4));
        g.fillPolygon(ai, ail, byte1);
        ai[0] = j;
        ail[0] = k;
        ai[1] = j + 11;
        ail[1] = k - k1;
        ai[2] = j + 11;
        ail[2] = k - 3 * k1;
        ai[3] = j;
        ail[3] = k - 11;
        g.setColor(new Color(c1, byte5, byte4));
        g.fillPolygon(ai, ail, byte1);
    }
}
}

```

//-----文件名 building.java, 程序编号 15.15-----

```

import javax.swing.*;
import java.awt.*;
public class building{
    JFrame mainFrame;
    houseCanvas palette;
    public building(){
        mainFrame=new JFrame("盖房子");
        palette=new houseCanvas();
        mainFrame.getContentPane().add(palette);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setSize(200,200);
        mainFrame.setVisible(true);
    }
    public static void main(String[] args) {
        new building();
    }
}

```

程序运行截图如图 15.11 所示。



图 15.11 程序运行截图

15.4 处 理 字 体

文字是程序与用户交互的最为重要的工具。目前的操作系统都支持多种字体，以满足程序美观的需要。在多数情况下，使用系统默认的字体以默认的形式显示，就可以完成基本的信息显示功能。但在某些情况下，为了程序的美观，还需要程序员自己处理字体。这就需要程序员掌握更多关于字体的基础知识以及处理技巧。

系统的字体有家族名（family name）、逻辑名（logical name）和外形名（face name）。家族名是字体的一般名称，如 Courier。逻辑名是指字体的类别，如 Monospaced。外形名是指特殊的字体，如 Courier Italic。

Java 中的字体处理，大多数情况下可以通过 Font 类中的方法来实现，下面就介绍这个类。

15.4.1 Font 类中的方法

Font 类中封装的方法很多，表 15.1 列出了其中一些常用的方法。

表 15.1 Font 中的常用方法

方 法	说 明
static Font decode(String str)	创建一个由str指定名称的字体对象
Map<TextAttribute,?> getAttributes()	获取当前字体对象的属性，以Map类型返回
String getFamily()	返回字体对象的家族名
static Font getFont(String nm)	返回由nm指定的系统属性相关的字体，如果该属性不存在，返回null
String getFontName()	返回当前字体对象的外形名
String getName()	返回当前字体对象的逻辑名
int getSize()	返回当前字体对象的尺寸，单位为像素

续表

方 法	说 明
int getStyle()	返回当前字体对象的样式值
boolean isBold()	如果字体包含黑体样式, 返回true, 否则返回false
boolean isItalic()	如果字体包含斜体样式, 返回true, 否则返回false
boolean isPlain()	如果字体包含纯文本样式, 返回true, 否则返回false

15.4.2 确定可用字体

在 14.12 节的字体对话框程序中, 曾经编程获得过系统字体的有关信息, 本节再来详细介绍一下如何使用 `GraphicsEnvironment` 类。在该类中定义了一个方法, 可以获取系统中安装的所有字体的家族名, 方法原型声明如下:

```
String [] getAvailableFontFamilyNames();
```

返回的家族名以字符串数组的形式存放。

除此之外, `GraphicsEnvironment` 类还定义了 `getAllFonts()` 方法, 它的原型声明如下:

```
Font [] getAllFonts();
```

该方法返回 `Font` 类型的数组, 系统中的每一个字体都以对象形式存储在该数组中。

这些方法都是 `GraphicsEnvironment` 类的实例成员, 需要先创建该类的对象, 但该类是抽象类, 不能直接使用它的构造方法。不过该类提供了一个静态方法, 可以获取本类对象的一个引用。它的原型声明如下:

```
static GraphicsEnvironment getLocalGraphicsEnvironment()
```

在下面的例子中, 演示了如何获取系统中安装的所有字体的家族名, 并将其显示在一个列表框中。

【例 15.13】 获取系统中安装的字体示例。

//-----文件名 GetFonts.java, 程序编号 15.16-----

```
import javax.swing.*;
import java.awt.*;

public class GetFonts{
    private JFrame mainJFrame;
    private JList nameList;
    private JScrollPane nameSPane;
    public GetFonts(){
        mainJFrame = new JFrame("获取系统字体");
        //获得 GraphicsEnvironment 类型的对象引用
        GraphicsEnvironment eq = GraphicsEnvironment.
            getLocalGraphicsEnvironment();
        //获取所有的字体家族名
        String[] availableFonts= eq.getAvailableFontFamilyNames();
        //存放到列表框中
        nameList=new JList(availableFonts);
```

```

nameSPane=new JScrollPane(nameList);
mainJFrame.add(nameSPane);
mainJFrame.setSize(300,300);
mainJFrame.setVisible(true);
mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[]){
    new GetFonts();
}
}

```

程序运行截图如图 15.12 所示。



图 15.12 程序运行截图

15.4.3 创建和使用字体

为了使用一个新字体，必须首先构造一个 `Font` 类型的对象，`Font` 构造方法的原型声明如下：

```
Font(String name, int style, int size)
```

其中，参数 `name` 可以是字体的逻辑名或者外形名。在所有的 Java 环境下，都支持下列字体：Dialog、DialogInput、Sans Serif、Serif、Monospaced 和 Symbol。Dialog 是系统对话框使用的字体。如果不指明字体，则默认为 Dialog。也可以使用其他专用环境支持的字体，但是这些字体在其他环境下可能无法使用。

`style` 是字体的风格，比如，纯文本、黑体或者斜体等，它们由 `Font` 所定义的三个常量来描述：PLAIN、BOLD 和 ITALIC。它们可以组合使用，例如，`Font.BOLD | Font.ITALIC` 表示粗斜体。

`size` 是字体的尺寸，以像素为单位。

创建 `Font` 对象之后，为了在组件之中使用它，需要使用组件的 `setFont()` 方法。

在下面的例子中，用户可以指定字体的名称、大小和风格，程序会将它们组合成一个新的字体对象，并在标签中显示出来。

【例 15.14】 创建和使用字体示例。

//-----文件名 ShowFonts.java, 程序编号 15.17-----

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ShowFonts implements ActionListener{
    private JFrame mainJFrame;
    private JComboBox nameBox, styleBox;
    private JTextField sizeText;
    private JLabel fontLabel;
    private JButton showBtn;
    private JPanel panell;

    public ShowFonts(){
        mainJFrame = new JFrame("显示指定字体");
        //显示系统可用字体
        GraphicsEnvironment eq = GraphicsEnvironment.
            getLocalGraphicsEnvironment();
        String[] availableFonts= eq.getAvailableFontFamilyNames();
        nameBox=new JComboBox(availableFonts);
        nameBox.setEditable(false);
        nameBox.setSelectedItem("宋体");
        //显示字体风格由用户选择
        String [] style={"正常","粗体","斜体","粗斜体"};
        styleBox = new JComboBox(style);
        styleBox.setEditable(false);
        //由用户输入想要的字体尺寸
        sizeText = new JTextField("12");
        sizeText.setColumns(4);
        //标签用于显示用户选择的字体
        fontLabel = new JLabel("字体示例");
        //创建按钮并安装监听器
        showBtn = new JButton("显示字体");
        showBtn.addActionListener(this);
        //在窗口中排列组件
        panell = new JPanel();
        panell.setLayout(new FlowLayout());
        panell.add(nameBox);
        panell.add(styleBox);
        panell.add(sizeText);
        mainJFrame.add(panell,BorderLayout.NORTH);
        mainJFrame.add(fontLabel,BorderLayout.CENTER);
        mainJFrame.add(showBtn,BorderLayout.SOUTH);
        mainJFrame.setSize(300,300);
        mainJFrame.setVisible(true);
        mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e){
        //分别获取用户选择输入的字体信息

```

```

int styleIndex = styleBox.getSelectedIndex();
String fontStr = (String)nameBox.getSelectedItemAt();
int fontSize = Integer.parseInt(sizeText.getText());
//组合成字体对象
Font userFont=new Font(fontStr,styleIndex,fontSize);
//为标签设置新的字体并显示
fontLabel.setFont(userFont);
}
public static void main(String args[]){
    new ShowFonts();
}
}

```

程序运行截图如图 15.13 所示。

15.4.4 自行管理字体

前面使用了字体、字型和大小来调整文字显示的效果。但是如果使用组件的 `setFont()` 方法来显示这样的文字，它对整个组件上的文字显示都会产生影响。也就是说，同一组件上的所有文字都是同一样式。而像 word 这样的文字编辑工具，允许在同一组件上显示不同样式的文字，要做到这一点，必须用 `Graphics` 中的 `drawString()` 方法在画布上将文字画出来。不过，这又产生了另外一个问题，就是如何知道某种字体所占据画面的大小，以便计算文字显示的位置。要解决这一问题，就必须使用 `FontMetrics` 自行管理字体。

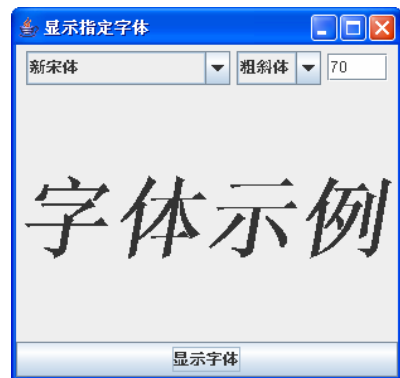


图 15.13 程序运行截图

在 `FontMetrics` 类中，封装了各种方法可以获取字体的相关信息，它们用下面这些术语来描述。

- ❑ **Height:** 某种字体中最高的字符从顶到底的高度。
- ❑ **Baseline:** 字符底端对齐的线，即基线。
- ❑ **Ascent:** 从基线到字符顶端的距离。
- ❑ **Descent:** 从基线到字符底部的距离。
- ❑ **Leading:** 一行文本底部到下一行文本顶端之间的距离。

用 `drawString()` 画出字符的过程中，它是以当前的字体和颜色在指定的位置输出一个字符串。不过这个位置是字符基线的左边缘，而不是像其他画图方法那样是指定的左上角。例如，如果在 (0, 0) 位置画出一个矩形，将看到一个完整的矩形；而如果在同一位置输出字符串 “Typesetting”，仅看见字符 y、p 和 g 的下半截。所以必须根据字体的外型来决定字符串应该显示的位置。

`FontMetrics` 定义了多种处理文本输出的方法，帮助程序将字符输出到窗口中合适的位置。表 15.2 列出了其中常用的方法。

表 15.2 FontMetrics中的常用方法

方 法	说 明
int bytesWidth(byte[] data, int off, int len)	以当前字体基线为准，计算要显示的字符串所占位置的像素总长度，off是字符串data中的起始位置，len是字符串的长度
int charsWidth(char[] data, int off, int len)	同上，不过参数是字符型
int charWidth(char ch)	以当前字体为准，计算字符ch所占的像素长度
int getAscent()	返回字体中大多数文字从基线到头部的高度，某些文字的高度可能超过这个值
int getDescent()	返回字体中大多数文字从基线到底部的高度，某些文字的高度可能超过这个值
Font getFont()	返回当前对象使用的字体
int getHeight()	返回以当前字体显示的文字需要占用的高度，它等于getAscent() + getDescent() + getLeading()值
int getLeading()	返回以当前字体计算的行间距
int getMaxAdvance()	返回当前字体中最宽的那个文字的基线宽度
int getMaxAscent()	返回当前字体中最高的那个文字从基线到头部的高度
int getMaxDescent()	返回当前字体中最高的那个文字从基线到底部的高度
int stringWidth(String str)	以当前字体基线为准，计算要显示的字符串所占位置的像素总长度

在下面的例子中，使用 drawString()在画布上画出各种字体，并且以右对齐的方式输出，这需要使用 FontMetrics 来计算字符串输出的位置。

【例 15.15】 自行管理字体示例。

//-----文件名 FontsCanvas.java，程序编号 15.18-----

```
//定义显示文字的画布
import javax.swing.*;
import java.awt.*;

public class FontsCanvas extends JPanel{
    private String msg;
    public FontsCanvas(String s){
        msg = s;
        setBackground(Color.white);
    }
    public FontsCanvas(){
        this("自行管理字体示例");
    }
    public void paintComponent(Graphics g){
        int maxWidth = getWidth();//获取画布的宽度
        int showX;                //文字输出的横坐标位置
        int showY = 0;            //文字输出的纵坐标位置
        int descent = 0;          //文字下半部所占位置
        int ascent = 0;           //文字上半部所占位置
        int leading = 0;          //行间距
        int totalWidth;           //字符串所占宽度
        FontMetrics fm;           //用于自行管理字体
        Font myFonts [] = new Font[4];
        //创建不同的字体准备显示
```

```

myFonts[0] = new Font("宋体", Font.PLAIN,12);
myFonts[1] = new Font("仿宋_GB2312", Font.BOLD,24);
myFonts[2] = new Font("黑体", Font.ITALIC,48);
myFonts[3] = new Font("楷体_GB2312", Font.ITALIC | Font.BOLD,60);
//用上述 4 种不同字体显示同一个字符串, 右对齐
for(int i=0;i<myFonts.length; ++i){
    g.setFont(myFonts[i]);
    fm = g.getFontMetrics();
    totalWidth = fm.stringWidth(msg);
    showX = maxWidth - totalWidth;
    ascent = fm.getMaxAscent();
    showY = showY + descent + ascent + leading;
    descent = fm.getMaxDescent();
    leading = fm.getLeading();
    g.drawString(msg,showX,showY);
}
}
}

```

然后要做的事情是将画布显示在窗口中, 程序如下:

//-----文件名 ManageFonts.java, 程序编号 15.19-----

```

import javax.swing.*;
import java.awt.*;
public class ManageFonts{
    private JFrame mainJFrame;
    private FontsCanvas palette; //可以显示多种文字的画布
    public ManageFonts(){
        mainJFrame = new JFrame("自行管理字体示例");
        palette = new FontsCanvas();
        mainJFrame.add(palette);
        mainJFrame.setSize(500,250);
        mainJFrame.setVisible(true);
        mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]){
        new ManageFonts();
    }
}

```

程序运行截图如图 15.14 所示。

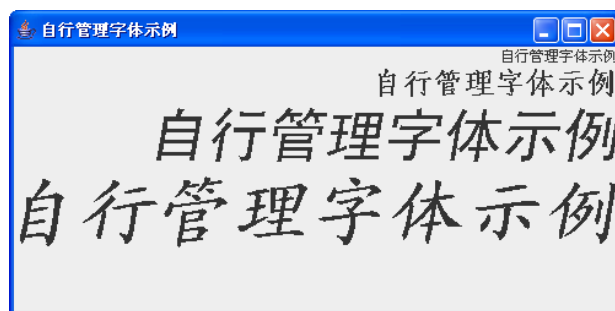


图 15.14 程序运行截图

15.4.5 字体特效显示

从本质上说，显示的文字也是图形，所以用在图形上的一些特效也可以用于文字，下面的例子演示了光照在文字上的效果。

所谓光照效果，就是在同一幅图片上用不同的颜色来显示其中的一个局部，这个局部的形状可以是任意的。对于光照效果，这个局部形状多数是圆形或者矩形。其中，矩形比圆形更简单，本例用的就是矩形。为了在局部中显示文字或图片，需要用到 `Graphics` 中的一个方法：

```
clipRect(int x, int y, int width, int height)
```

该方法被称为裁剪函数，它可以设置一个矩形区域，以后所有显示的文字或图像都在该区域中，超过该区域的部分将不会被画出。只要改变这个区域的位置，就可以看到光照移动的效果。

与其类似的方法还有：

```
setClip(int x, int y, int width, int height)
```

以及

```
setClip(Shape clip)
```

其中，最后一个方法的参数是 `Shape` 类型，它可以是圆形或者其他任意的多边形。

设置裁剪区域后，对所有的绘制方法都有效。如果需要该设置区域失效，可以使用 `setClip(null)`，就恢复成正常的绘图模式。

【例 15.16】 字体特效显示示例。

//-----文件名 LightingLiteral.java，程序编号 15.20-----

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.TimerTask;
import java.util.Timer;
public class LightingLiteral implements ActionListener{
    String title = "光照文字";           //显示的文字
    Font myFont = new Font("宋体", Font.BOLD, 48); //显示的字体
    JPanel palette;
    JFrame mainFrame;
    JButton startBtn;
    Container con;
    Timer myTimer;
    Refresh task;
    boolean startFlag;
    public LightingLiteral() {
        mainFrame = new JFrame(title);
        palette = new JPanel();
        startBtn = new JButton("开始");
```

```

startFlag = true;
startBtn.addActionListener(this);
con = mainFrame.getContentPane();
con.add(palette, BorderLayout.CENTER);
con.add(startBtn, BorderLayout.NORTH);
mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
mainFrame.setSize(300, 300);
mainFrame.setVisible(true);
}
public void actionPerformed(ActionEvent e){
    if (startFlag){
        myTimer=new Timer();
        task=new Refresh();
        myTimer.schedule(task,50,50);           //启动定时器，时间间隔 50 毫秒
        startBtn.setText("停止");
    }else{
        myTimer.cancel();
        myTimer = null;
        task = null;
        startBtn.setText("开始");
    }
    startFlag = !startFlag;
}
//用定时器来绘图
class Refresh extends TimerTask{
    int pos = 0;
    int blink_width = 20;                       //光条的宽度
    Graphics g = palette.getGraphics();          //注意画笔的获取方式
    FontMetrics myFM = g.getFontMetrics(myFont);
    int height = myFM.getHeight();               // 计算文字的高度
    int top = myFM.getAscent();
    int width = myFM.stringWidth(title);         //计算字符串的宽度
    public Refresh(){
        g.setFont(myFont);
    }
    public void run(){
        g.setColor(Color.blue);                 //用蓝色显示文字
        g.drawString(title, 0, top);             //第一遍显示
        g.clipRect(pos, 0, blink_width, height); //设置裁剪区域
        g.setColor(Color.yellow);               //用黄色显示文字
        g.drawString(title, 0, top); //第二遍显示，它只会显示在裁剪区域中
        pos = (pos + 5) % width;                //移动裁剪区域的位置
        g.setClip(null);                        //让裁剪区域失效，准备重新绘制蓝色文字
    }
}
public static void main(String[] args){
    new LightingLiteral();
}
}

```

程序运行截图如图 15.15 所示。

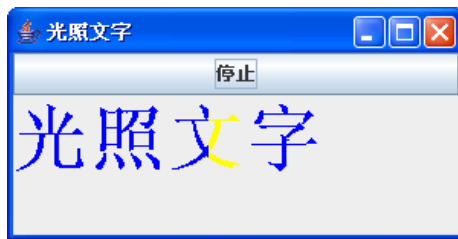


图 15.15 程序运行截图

其中，黄色光条是可以移动的，但截图上效果不明显。

15.5 图像的显示

本节来介绍如何处理和显示图像。图像比图形更为复杂，它们是以像素为单位进行描述的。

在 Java 中显示图像的方法通常有两种：一种是用标准的控件加载并显示。java 中的大多数可视控件都是可以直接显示图像的，例如，JLabel、JButton 等，这既可以美化程序界面，同时还减轻了程序员的负担。另外一种方法是用 Graphics 类中的 drawImage() 方法，该方法有多个重载版本，以满足不同的需要。第一种方法最为简单，而且不用程序员操心窗口的重绘问题，但是速度较慢，而第二种方法则恰好相反。具体选择哪一种方法，要看实际应用情况而定。一般如果只是静态显示图片，可以选择第一种方法，否则，应该选择第二种方法。

不过，在某些情况下，光使用这些控件的图像显示功能不能满足需要，就需要用更为低级、功能更强的处理图像的类。Java 中主要通过 Image 及其子类来处理图像，默认可处理的图像格式有：png、jpeg 和 gif，但是不包括 bmp 格式。如果是 ico 格式，则需要 ImageIcon 类来处理。

除了 Image 类外，另外还需要一些辅助的类，例如：ImageFilter、ImageProducer、FilteredImageSource 等。这些类非常之多，都定义在 java.awt.image 包中，这里不可能一一介绍它们。下面通过一些例子来演示如何使用它们。

15.5.1 标准的图像显示

显示一幅完整图片，最简单的方法是用 JLabel 来加载并显示它，方法声明原型如下：

```
void setIcon(Icon icon)
```

它的参数是 Icon 类型，而 Icon 是一个接口。因此，在实际应用中，需要使用它的子类：ImageIcon，用它来创建一个 Icon 对象加载到标签中。

【例 15.17】 用标签显示图像示例。

//-----文件名 viewPic.java，程序编号 15.21-----

```

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.File;
public class viewPic implements ActionListener{
    JLabel imgLabel;
    JFrame mainJframe;
    Container con;
    JTextField fileField;
    JButton openBtn;
    JPanel pane;
    JScrollPane spanel;
    ImageIcon img;
    public viewPic() {
        mainJframe=new JFrame("图像显示示例");
        con=mainJframe.getContentPane();
        pane=new JPanel();
        pane.setLayout(new FlowLayout());
        openBtn= new JButton("打开文件");
        openBtn.addActionListener(this);
        fileField = new JTextField();
        fileField.setColumns(20);
        pane.add(fileField);
        pane.add(openBtn);
        imgLabel = new JLabel();
        spanel = new JScrollPane(imgLabel);
        con.add(pane,BorderLayout.NORTH);
        con.add(spanel,BorderLayout.CENTER);
        mainJframe.setSize(400,400);
        mainJframe.setVisible(true);
        mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e) {
        try{
            JFileChooser chooser = new JFileChooser();
            if(chooser.showOpenDialog(mainJframe)==JFileChooser.
            APPROVE_OPTION){
                File tempfile= chooser.getSelectedFile();
                fileField.setText(tempfile.toString()) ;
                //加载图片到内存中,同时创建 Icon 对象
                img = new ImageIcon(fileField.getText());
                //将图片显示在标签中
                imgLabel.setIcon(img);
            }
        }catch(Exception e1){
            JOptionPane.showMessageDialog(mainJframe,"无法加载图片!");
        }
    }
    public static void main(String[] args) {
        new viewPic();
    }
}

```

程序运行截图如图 15.16 所示。



图 15.16 程序运行截图

这个程序非常简单，无需过多解释。不过本节稍后的程序将在此程序的基础上逐步完善，写成一个小小的图片浏览器。

15.5.2 显示局部图像

用 JLabel 显示大型图片时速度会比较慢。有时候只需要看到这幅图片的一部分，这可以对图片进行裁剪。裁剪一幅图片要按照下列步骤来进行。

(1) 获取原始图片数据。在程序 15.21 中，原始图片存储在 ImageIcon 对象 img 中，利用它的 getImage() 方法可以获取一个 Image 对象，该对象有一个 getSource() 方法，可以获取原始图片数据。

(2) 设置要裁剪的区域。这需要使用 CropImageFilter 对象创建一个虚拟的裁剪区域出来。

(3) 利用 FilteredImageSource 对象从原始图片数据和指定的裁剪区域中取出新的图片数据。

(4) 利用 Toolkit.getDefaultToolkit().createImage() 方法从 FilteredImageSource 对象中生成新的图片。

(5) 将图片显示在 JLabel 中。

这个过程比较复杂，需要用到多种图像处理类，并且要进行多次转换，下面提供了示例函数。将此函数添加到程序 15.21 中，并增加相应的按钮，就可以使用了。为了节省篇幅，这里以及随后的两个例子不再重复界面设计部分，而只提供相应的处理函数。

【例 15.18】 显示局部图像示例。

//-----文件名 viewPic.java (部分)，程序编号 15.22-----

```
//裁剪图像
private void cutfile(){
    ImageFilter cropFilter;
    Image croppedImage;
    cropFilter =new CropImageFilter(100,100,200,200); //四个参数分别为图像起
    点坐标和宽高
    ImageProducer producer;
    producer=new FilteredImageSource(img.getImage().getSource(),
    cropFilter);
    croppedImage=Toolkit.getDefaultToolkit().createImage(producer);
    imgLabel.setIcon(new ImageIcon(croppedImage));
}
```

程序运行截图如图 15.17 所示。

15.5.3 图像缩放

图像的缩放比较简单,因为在 `Image` 类中就提供了一个 `getScaledInstance()` 方法,可以用来缩放图像。它的原型声明如下:

```
Image getScaledInstance(int width, int height,
int hints)
```

它可以创建一个当前图像的缩放版本并以 `Image` 的形式返回。其中, `width` 和 `height` 是新图像的宽和高。`Hints` 标志缩放时所采用的算法,可以是下列常量之一: `SCALE_AREA_AVERAGING`、`SCALE_DEFAULT`、`SCALE_FAST`、`SCALE_REPLICATE` 和 `SCALE_SMOOTH`。

下面的函数演示了如何实现图像的缩放,它仍然是 `viewPic` 文件的一部分。

【例 15.19】 缩放图像示例。

//-----文件名 viewPic.java (部分), 程序编号 15.23-----

```
//本方法实现缩放图像功能, multipe 是缩放的倍数。大于 1 表示放大, 小于 1 为缩小
private void scalefile(double multipe){
    Image sourceImg=img.getImage();
    Image scaledImg;
    scaledImg=sourceImg.getScaledInstance((int)(sourceImg.getWidth
    (null)*multipe),
    (int)(sourceImg.getHeight(null)*multipe),
    Image.SCALE_DEFAULT);
    imgLabel.setIcon(new ImageIcon(scaledImg));
}
```

程序运行中, 缩小显示图像截图如图 15.18 所示。



图 15.17 程序运行截图

15.5.4 灰度变换

灰度变换可以将彩色图片以黑白的形式显示出来。常用的灰度变换公式有两种, 一种

是如下公式：

$$C = R*0.3 + G*0.59 + B*0.11$$



图 15.18 程序运行截图

其中，R、G、B 分别是像素的三个三原色分量，得到的结果就是像素的灰度值。
另外一种方法是如下公式：

$$C = \max\{R, G, B\}$$

即以三原色中最大值为像素的灰度值。

灰度变换的算法很简单，但在 Java 中实现起来却有一定的难度。这是由于 Image 也是采用 MVC 模式构建的，应用程序无法直接访问其中的像素值，需要程序员继承其 Model 和 Filter，在 Model 中修改像素值。

下面的例子中演示了如何构建一个自己的 Model 和 Filter 来实现灰度变换。

【例 15.20】 灰度变换示例。

//-----文件名 GrayModel.java，程序编号 15.24-----

```
//实现一个具备灰度变换功能的 Model
import java.awt.image.ColorModel;
class GrayModel extends ColorModel{
    ColorModel sourceModel;
    public GrayModel(ColorModel sourceModel) {
        super(sourceModel.getPixelSize());
        this.sourceModel=sourceModel;
    }
    public int getAlpha(int pixel) {
        return sourceModel.getAlpha(pixel);
    }
    public int getRed(int pixel) {
        return getGrayLevel(pixel);
    }
    public int getGreen(int pixel) {
```

```

        return getGrayLevel(pixel);
    }
    public int getBlue(int pixel) {
        return getGrayLevel(pixel);
    }
    //修改像素值，实现灰度变换
    protected int getGrayLevel(int pixel) {
        return (int)(sourceModel.getRed(pixel)*0.3 +
                    sourceModel.getGreen(pixel)*0.59 +
                    sourceModel.getBlue(pixel)*0.11 );
    }
}

```

接下来再实现自己的 GrayFilter 类。

//-----文件名 GrayFilter.java，程序编号 15.25-----

```

import java.awt.image.*;
public class GrayFilter extends RGBImageFilter {
    public GrayFilter() {
        canFilterIndexColorModel = true;
    }
    //设置 Model 为自己编制的 GrayModel
    public void setColorModel(ColorModel cm) {
        substituteColorModel(cm, new GrayModel(cm));
    }
    public int filterRGB(int x, int y, int pixel) {
        return pixel;
    }
}

```

下面是使用这两个类，创建一个 Image，然后在 Label 中显示出来。

//-----文件名 viewPic.java（部分），程序编号 15.26-----

```

private void grayfile(){
    GrayFilter filter;
    Image gray;
    ImageProducer producer;
    filter=new GrayFilter();
    //用 GrayFilter 来创建 Image
    producer=new FilteredImageSource(img.getImage().getSource(),filter);
    gray=Toolkit.getDefaultToolkit().createImage(producer);
    imgLabel.setIcon(new ImageIcon(gray));
}

```

图 15.19 是实现灰度变换之后的截图。

15.5.5 一个简单的图片浏览器

有了前面 4 小节的基础，将这些程序合成在一起，就可以成为一个简单的图片浏览器。下面是完整的源程序，其中用到的 GrayFilter 和 GrayModel 15.5.4 节已经提供，不再重复。



图 15.19 程序运行截图

【例 15.21】 一个简单的图片浏览器

//-----文件名 viewPic.java（完整），程序编号 15.27-----

```
import javax.swing.*;
import java.awt.image.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;

public class viewPic implements ActionListener{
    JLabel imgLabel;
    JFrame mainJframe;
    Container con;
    JTextField fileField;
    JButton openBtn,cutBtn,shrinkBtn,zoomBtn,grayBtn;
    JPanel pane;
    JScrollPane spane;
    ImageIcon img;
    public viewPic() {
        mainJframe=new JFrame("图像显示示例");
        con=mainJframe.getContentPane();
        pane=new JPanel();
        pane.setLayout(new FlowLayout());
        openBtn= new JButton("打开文件");
        openBtn.addActionListener(this);
        cutBtn=new JButton("显示部分");
        cutBtn.addActionListener(this);
        shrinkBtn=new JButton("缩小图像");
        shrinkBtn.addActionListener(this);
        zoomBtn=new JButton("放大图像");
        zoomBtn.addActionListener(this);
```

```

        grayBtn=new JButton("灰度变换");
        grayBtn.addActionListener(this);
        fileField = new JTextField();
        fileField.setColumns(20);
        pane.add(fileField);
        pane.add(openBtn);
        pane.add(cutBtn);
        pane.add(shrinkBtn);
        pane.add(zoomBtn);
        pane.add(grayBtn);
        imgLabel=new JLabel();
        con.add(pane,BorderLayout.NORTH);
        spane = new JScrollPane(imgLabel);
        con.add(spane,BorderLayout.CENTER);
        mainJframe.setSize(400,400);
        mainJframe.setVisible(true);
        mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==openBtn)
            openfile();
        else if(e.getSource()==cutBtn)
            cutfile();
        else if(e.getSource()==shrinkBtn)
            scalefile(0.5);
        else if(e.getSource()==zoomBtn)
            scalefile(2.0);
        else if(e.getSource()==grayBtn)
            grayfile();
    }
    public static void main(String[] args) {
        new viewPic();
    }
    //打开文件并显示
    private void openfile(){
        try{
            JFileChooser chooser = new JFileChooser();
            if(chooser.showOpenDialog(mainJframe)==JFileChooser.
            APPROVE_OPTION){
                File tempfile= chooser.getSelectedFile();
                fileField.setText(tempfile.toString());
                img = new ImageIcon(fileField.getText());
                imgLabel.setIcon(img);
            }
        }catch(Exception e){
            JOptionPane.showMessageDialog(mainJframe,"无法加载图片!");
        }
    }
    //显示图片局部
    private void cutfile(){
        ImageFilter cropFilter;

```

```

        Image croppedImage;
        cropFilter =new CropImageFilter(100,100,200,200);
        ImageProducer producer;
        producer=new FilteredImageSource(img.getImage().getSource(),
        cropFilter);
        croppedImage=Toolkit.getDefaultToolkit().createImage(producer);
        imgLabel.setIcon(new ImageIcon(croppedImage));
    }
    //缩放图片
    private void scalefile(double multipe){
        Image sourceImg=img.getImage();
        Image scaledImg;
        scaledImg=sourceImg.getScaledInstance(
            (int)(sourceImg.getWidth(null)*multipe),
            (int)(sourceImg.getHeight(null)*multipe),
            Image.SCALE_DEFAULT);
        imgLabel.setIcon(new ImageIcon(scaledImg));
    }
    //显示灰度图片
    private void grayfile(){
        GrayFilter filter;
        Image gray;
        ImageProducer producer;
        filter=new GrayFilter();
        producer=new FilteredImageSource(img.getImage().getSource(),
        filter);
        gray=Toolkit.getDefaultToolkit().createImage(producer);
        imgLabel.setIcon(new ImageIcon(gray));
    }
}

```

程序运行情况在前面都已经截图，这里不再重复。

15.5.6 合成两幅图片

在某些情况下，可能需要将两幅图片合成在一起显示。典型的例子是将一幅图片作为水印嵌入到另外一幅图片中。利用 15.3.3 小节介绍的 **alpha** 值可以比较容易地实现这一任务。

为了实现水印效果，有两种做法。一是先以正常模式显示水印图片，然后设置 **alpha** 值，这个值应该比较接近 1.0，通常在 0.8~0.95 之间，最后再显示前景图片。另外一种做法是先以正常模式显示前景图片，然后设置 **alpha** 值，这个值比较接近 0，通常在 0.05~0.2 之间，最后再显示水印图片。

两种方法的效果差不多，不过，作为水印的图片最好是背景透明的 gif 格式的图片，这样的水印效果对前景图片影响最小。

【例 15.22】 合成两幅图片。

首先要做的事情是编写自己的画布，所要显示的图片在此画布上显示。

//-----文件名 CombinerCanvas.java, 程序编号 15.28-----

```
import javax.swing.*;
import java.awt.*;

public class CombinerCanvas extends JPanel {
    private Image foreImg = null, backImg = null;
    private float _alpha = 1.0f;
    //三个参数依次为: 前景图片、背景图片(水印)、透明度
    public CombinerCanvas(Image forePic, Image backPic, float alpha){
        foreImg = forePic;
        backImg = backPic;
        _alpha = alpha;
    }
    //创建一个指定 alpha 值的 AlphaComposite 对象
    private AlphaComposite makeComposite(float alpha) {
        int type = AlphaComposite.SRC_OVER;
        return(AlphaComposite.getInstance(type, alpha));
    }
    //用指定的 alpha 值来绘制前景和背景图片
    private void combinImage(Graphics2D g2d) {
        Composite originalComposite = g2d.getComposite();
        //用默认透明度绘制前景图片
        g2d.drawImage(foreImg, 0, 0, this);
        //设置透明度, 准备绘制水印图片
        g2d.setComposite(makeComposite(_alpha));
        g2d.drawImage(backImg, 0, 0, this);
        //将透明度设置回默认的模式
        g2d.setComposite(originalComposite);
    }
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        combinImage((Graphics2D)g);
    }
}
```

然后编写测试程序, 使用该画布来合成图片。

//-----文件名 CombinPic.java, 程序编号 15.29-----

```
import javax.swing.*;
import java.awt.*;

public class CombinPic{
    JFrame mainFrame;
    CombinerCanvas palette;
    JScrollPane spane;
    public CombinPic(Image fore, Image back, float alpha){
```

```

mainFrame=new JFrame("水印图片示例");
palette=new CombinerCanvas(fore,back,alpha);
spane = new JScrollPane(palette);
mainFrame.add(spane, BorderLayout.CENTER);
mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
mainFrame.setSize(400,400);
mainFrame.setVisible(true);
try{
    Thread.sleep(1000);
    palette.repaint();
}catch(Exception e1) {}
}
public static void main(String[] args) {
    if (args.length<2 || args.length>3){
        System.err.println("调用格式不对, 应该是: java CombinPic 前景图片名
        背景图片名");
        System.err.println("或者: java CombinPic 前景图片名 背景图片名 透明度");
        System.err.println("透明度的值从 0.0-1.0, 值越小, 水印越淡。");
        System.err.println("作为水印的图片最好是背景透明的 gif 图片");
        System.err.println("例如: java CombinPic myPic.jpg back.gif 0.1");
        return ;
    }
    Image fore = Toolkit.getDefaultToolkit().createImage(args[0]);
    Image back = Toolkit.getDefaultToolkit().createImage(args[1]);
    if (fore == null){
        System.err.println("第一个文件不是可以识别的图片");
        return ;
    }
    if (back == null){
        System.err.println("第二个文件不是可以识别的图片");
        return ;
    }
    if (args.length==2)
        new CombinPic(fore,back,0.1f);
    else{
        float alpha = Float.parseFloat(args[2]);
        new CombinPic(fore,back,alpha);
    }
}
}

```


在命令行输入:

```
java CombinPic sun1.jpg back.jpg 0.1
```

之后, 程序就可以正常运行了。程序运行截图如图 15.20 所示。



图 15.20 程序运行截图

提示：调整命令行中的 alpha 值，可以使得水印效果更浓或更淡。

15.5.7 光照特效

在 15.4.5 小节中，曾经演示过文字上的光照效果，本节来编写一个更为复杂的程序，演示光照在图像上的效果。

要实现光照效果，仍然要使用裁剪区域，这里将裁剪区域设置为圆形。将要显示的图片在裁剪区域中的部分显示出来，其余部分填充为深灰色，形成对比效果。缓慢移动裁剪区域，就可以形成光斑在移动的效果。基本的方法与文字上的光照效果相同，不过移动的算法更复杂一些。

【例 15.23】 光照特效示例。

//-----文件名 illumination.java，程序编号 15.30-----

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.TimerTask;
import java.util.Timer;
public class illumination implements ActionListener {
    JPanel palette;
    JFrame mainFrame;
    JButton startBtn;
    Container con;
    Timer myTimer;
    Refresh task;
    boolean startFlag;
```



```

Image img;
int width,height;

public illumination(String imgfile) {
    mainFrame = new JFrame("光照特效示例");
    palette = new JPanel();
    startBtn = new JButton("开始");
    startFlag = true;
    startBtn.addActionListener(this);
    con = mainFrame.getContentPane();
    con.add(palette, BorderLayout.CENTER);
    con.add(startBtn, BorderLayout.NORTH);
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    mainFrame.setSize(400,400);
    mainFrame.setVisible(true);
    width = palette.getWidth();
    height = palette.getHeight();
    img = Toolkit.getDefaultToolkit().createImage(imgfile);
}

public void actionPerformed(ActionEvent e){
    if (startFlag){
        myTimer=new Timer();
        task=new Refresh();
        //等待一下, 让用户看清楚原始图片
        try{
            Thread.sleep(2000);
        }catch(Exception el){}
        myTimer.schedule(task,100,100); //启动定时器, 时间间隔 100 毫秒
        startBtn.setText("停止");
    }else{
        myTimer.cancel();
        myTimer = null;
        task = null;
        startBtn.setText("开始");
    }
    startFlag = !startFlag;
}

public static void main(String[] args){
    if (args.length!=1){
        System.err.println("命令格式错误, 你必须指定要显示的图片。");
        System.err.println("例如: java illumination test.jpg");
    }else{
        new illumination(args[0]);
    }
}

//定义一个内部类作为定时器, 用它来移动光斑
class Refresh extends TimerTask{
    int i,j,k; //i,j 是光斑的坐标, k 是光斑当前的方向
    int dx, dy; //光斑每次移动的距离
    int esize; //光斑的大小
    public Refresh(){

```

```
i = (int)((Math.random() * (double)width) / 2D);
j = (int)((Math.random() * (double)height) / 2D);
k = (int)(Math.random() * 3D);
dx = dy = 5;
esize = 100;
//显示原始图片
Graphics g = palette.getGraphics();
g.drawImage(img, 0, 0, width, height, palette);
}
public void run() {
    //计算光斑应该移动的位置
    switch(k){
        case 0:
            if(i < width - esize){
                i += dx;
            } else {
                k = 1;
            }
            if(j < height - esize){
                j += dy;
            } else {
                k = 3;
            }
            break;
        case 1:
            if(i > 0){
                i -= dx;
            } else {
                k = 0;
            }
            if(j < height - esize) {
                j += dy;
            } else {
                k = 2;
            }
            break;
        case 2:
            if(i > 0){
                i -= dx;
            } else {
                k = 3;
            }
            if(j > 0) {
                j -= dy;
            } else {
                k = 1;
            }
            break;
        case 3:
            if(i < width - esize){
                i += dx;
```

```

        } else {
            k = 2;
        }
        if(j > 0){
            j -= dy;
        } else {
            k = 0;
        }
        break;
    }
    //光斑位置计算完毕，在当前位置画出图片局部
    replace(i, j);
}
//在指定位置显示图片
public void replace(int i, int j) {
    Graphics g = palette.getGraphics();
    //用深灰色填充，造出黑暗效果
    g.setColor(Color.darkGray);
    g.fillRect(0, 0, width, height);
    //设置裁剪区域
    g.setClip(new Ellipse2D.Float(i,j, esize, esize));
    //在裁剪区域中绘出图像
    g.drawImage(img, 0, 0, width, height, palette);
    //恢复全局绘画
    g.setClip(null);
}
} //内部类结束
}

```

在命令行输入：

```
java illumination view.jpg
```

图 15.21 是原始的图片。图 15.22 是运行中的截图。



图 15.21 原始图片

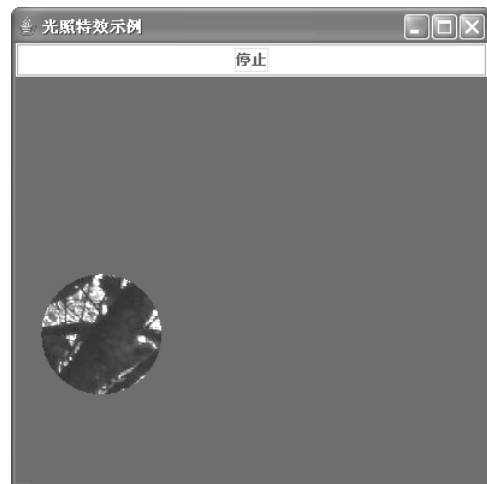



图 15.22 程序运行中截图

这个光斑会像小球一样移动，碰到窗口的四壁就会弹回来。

15.6 视频文件的播放

视频是多媒体数据中最为复杂，也是最为重要的一类数据。标准的 jdk 不支持视频文件的解码和播放，这给 Java 程序员带来了很大的麻烦。为了弥补这一缺陷，Sun 公司专门提供了一个 JMF，帮助程序员解决视频和音频文件的解码和播放问题。本节将利用 JMF API 来实现自己的媒体播放器。

 **注意：**JMF 需要到 Sun 公司的网站下载，本书所附光盘的第 15 章里面也有该文件，名称为 JMF.EXE。

Java 媒体框架（JMF）能够帮助程序员编写出功能强大的多媒体程序，却不用关心底层复杂的实现细节。JMF API 的使用相对比较简单，但是能够满足几乎所有多媒体编程的需求——包括视频和音频。

JMF 目前的最新版本是 2.2，Sun 通过它向 Java 中引入处理多媒体的能力。下面是 JMF 所支持的功能的一个概述。

- ☐ 可以在 Java Applet 和 Application 程序中播放各种媒体文件。例如，AU、AVI、MIDI、MPEG、QuickTime 和 WAV 等文件。
- ☐ 可以播放从互联网上下载的媒体流。
- ☐ 可以利用麦克风和摄像机一类的设备截取音频和视频，并保存成多媒体文件。
- ☐ 处理多媒体文件，转换文件格式。
- ☐ 向互联网上传音频和视频数据流。
- ☐ 在互联网上广播音频和视频数据。

要使用 JMF，需要到 Sun 公司的网站下载 JMF，然后安装在本地机器上。安装好后，如果要在 eclipse 中使用它，还需要在搜索路径中引入它。

15.6.1 JMF 中的常用术语

JMF 的功能非常强大，使用起来也不麻烦，但需要程序员理解一些基本的概念。JMF 中的一些常见术语解释如下：

1. 数据源

就像 CD 中保存了歌曲一样，数据源中包含了媒体数据流。在 JMF 中，DataSource 对象就是数据源，它可以是一个多媒体文件，也可以是从互联网上下载的数据流。对于 DataSource 对象，一旦确定了它的位置和类型，对象中就包含了多媒体的位置信息和能够播放该多媒体的软件信息。当创建了 DataSource 对象后，可以将它送入 Player 对象中，而 Player 对象不需要关心 DataSource 中的多媒体是如何获得的，以及格式是什么。

在某些情况下，程序员需要将多个数据源合并成一个数据源。例如，当用户在制作一

段录像时，需要将音频数据源和视频数据源合并在一起。JMF 支持数据源合并。

2. 截取设备

截取设备指的是可以截取到音频或视频数据的硬件，如麦克风、摄像机等。截取到的数据可以被送入 Player 对象中进行处理。

3. 播放器

在 JMF 中对应播放器的接口是 Player。Player 对象将音频/视频数据流作为输入，然后将数据流输出到音箱或屏幕上，就像 CD 播放机读取 CD 唱片中的歌曲，然后将信号送到音箱上一样。Player 对象有多种状态，JMF 中定义了 JMF 的六种状态，在正常情况下 Player 对象需要经历每个状态，然后才能播放多媒体。下面是对这些状态的说明。

- ❑ **Unrealized**: 在这种状态下，Player 对象已经被实例化，但是并不知道它需要播放的多媒体的任何信息。
- ❑ **Realizing**: 当调用 realize() 方法时，Player 对象的状态从 Unrealized 转变为 Realizing。在这种状态下，Player 对象正在确定它需要占用哪些资源。
- ❑ **Realized**: 在这种状态下 Player 对象已经确定了它需要哪些资源，并且也知道需要播放的多媒体的类型。
- ❑ **Prefetching**: 当调用 prefectch() 方法时，Player 对象的状态从 Realized 变为 Prefetching。在该状态下的 Player 对象正在为播放多媒体做一些准备工作，其中包括加载多媒体数据，获得需要独占的资源等。这个过程被称为预取（Prefetch）。
- ❑ **Prefetched**: 当 Player 对象完成了预取操作后就到达了该状态。
- ❑ **Started**: 当调用 start() 方法后，Player 对象就进入了该状态并播放多媒体。

4. 处理器

处理器对应的接口是 Processor，它是一种播放器。在 JMF API 中，Processor 接口继承了 Player 接口。Processor 对象除了支持 Player 对象支持的所有功能，还可以控制对于输入的多媒体数据流进行何种处理，以及通过数据源向其他的 Player 对象或 Processor 对象输出数据。

除了在播放器中提到了六种状态外，Processor 对象还包括两种新的状态，这两种状态是在 Unrealized 状态之后，但是在 Realizing 状态之前。

- ❑ **Configuring**: 当调用 configure() 方法后，Processor 对象进入该状态。在该状态下，Processor 对象连接到数据源并获取输入数据的格式信息。
- ❑ **Configured**: 当完成数据源连接，获得输入数据格式的信息后，Processor 对象就处于 Configured 状态。

5. 数据格式

Format 对象中保存了多媒体的格式信息。该对象中本身没有记录多媒体编码的相关信息，但是它保存了编码的名称。Format 的子类包括 AudioFormat 和 VideoFormat 类，VideoFormat 又有六个子类：H261Format、H263Format、IndexedColorFormat、JPEGFormat、

RGBFormat 和 YUVFormat 类。

6. 管理器

JMF 提供了下面 4 种管理器：

- ❑ **Manager:** Manager 相当于两个类之间的接口。例如，当你需要播放一个 DataSource 对象，你可以通过使用 Manager 对象创建一个 Player 对象来播放它。使用 Manager 对象可以创建 Player、Processor、DataSource 和 DataSink 对象。
- ❑ **PackageManager:** 该管理器中保存了 JMF 类的注册信息。
- ❑ **CaptureDeviceManager:** 该管理器中保存了截取设备的注册信息。
- ❑ **PlugInManager:** 该管理器中保存了 JMF 插件的注册信息。

15.6.2 播放器实例

本小节编写一个简单的媒体播放器。读者将看到，只要理解了前面介绍的概念，使用 JMF 编写它是比较简单的事情。

下面是一个使用 Player 来播放多媒体的例子，为了使用它，需要做好以下几件事情：

- ❑ 创建 Player 对象。这可以通过 Manager 类的 createPlayer() 方法创建 Player 对象。Manager 对象使用多媒体的 URL 或 MediaLocator 对象来创建 Player 对象。
- ❑ 创建一个图像显示部件，使得播放的视频能在它上面显示出来。这要通过调用 getVisualComponent() 方法得到 Player 对象的图像部件。
- ❑ 创建一个控制面板（当然，这个不是必须的），在上面可以控制媒体的播放、停止和暂停等。编程时只要用 getControlPanelComponent() 创建就可以了，然后不需要对其编写任何程序。
- ❑ 继承 ControllerListener 接口，实现其中的 controllerUpdate 方法。一旦 Player 创建成功，该事件就会被触发，表示可以开始播放了。

【例 15.24】 媒体播放器编写示例。

//-----文件名 playVideo.java，程序编号 15.31-----

```
import javax.swing.*;
import javax.media.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
//本类必须实现 ControllerListener 接口
public class playVideo implements ControllerListener, ActionListener {
    protected JTextField fileField;
    protected JButton openBtn, startBtn, pauseBtn, resumBtn, stopBtn;
    protected Container con;
    protected JPanel pane;
    //这两个部件分别用来显示视频图像和播放控件
    protected Component visualComp=null, controlComp=null;
    protected JFrame mainJframe;
    protected Player player=null;
```

```
public playVideo() {
    mainJframe=new JFrame("播放视频");
    con=mainJframe.getContentPane();
    pane=new JPanel();
    pane.setLayout(new FlowLayout());
    fileField=new JTextField();
    fileField.setColumns(25);
    openBtn=new JButton("选择");
    startBtn=new JButton("开始");
    pauseBtn=new JButton("暂停");
    resumBtn=new JButton("继续");
    stopBtn=new JButton("停止");
    openBtn.addActionListener(this);
    startBtn.addActionListener(this);
    pauseBtn.addActionListener(this);
    resumBtn.addActionListener(this);
    stopBtn.addActionListener(this);
    pane.add(fileField);
    pane.add(openBtn);
    pane.add(startBtn);
    pane.add(pauseBtn);
    pane.add(resumBtn);
    pane.add(stopBtn);
    con.add(pane,BorderLayout.NORTH);
    mainJframe.setSize(600,500);
    mainJframe.setVisible(true);
    mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public static void main(String[] args) {
    new playVideo();
}

public void actionPerformed(ActionEvent arg0) {
    Object obj;
    obj=arg0.getSource();
    try{
        if(obj==openBtn){
            openfile();
        }else if(obj==startBtn){
            if (player!=null)
                player.close();
            player=Manager.createPlayer(
                new MediaLocator("file:///"+fileField.getText()));
            player.addControllerListener(this);
            player.start();
        }else if(obj==pauseBtn){
            if (player!=null) player.stop();
        }else if(obj==resumBtn){
            if(player!=null) player.start();
        }else if(obj==stopBtn){
            if(player!=null){
                player.close();
            }
        }
    }
}
```

```

        player=null;
    }
}
}catch(Exception e){
    JOptionPane.showMessageDialog(mainJframe,"无法播放文件! ");
}
}
//准备播放视频文件
public void controllerUpdate(ControllerEvent e) {
    if (e instanceof RealizeCompleteEvent) {
        //清除上次播放的残余
        if(visualComp!=null)
            con.remove(visualComp);
        if(controlComp!=null)
            con.remove(controlComp);
        //重新设置本次播放的视频图像显示控件
        if (( visualComp = player.getVisualComponent())!=null)
            con.add(visualComp, BorderLayout.CENTER);
        //重新设置本次的控制控件
        if ((controlComp = player.getControlPanelComponent()) != null)
            con.add(controlComp, BorderLayout.SOUTH);
        mainJframe.validate();
    }
}
//让用户选择视频文件
private void openfile(){
    JFileChooser chooser = new JFileChooser();
    if(chooser.showOpenDialog(mainJframe)==JFileChooser.APPROVE_OPTION){
        File tempfile= chooser.getSelectedFile();
        fileField.setText(tempfile.toString());
    }
}
}

```


 注意：读者务必正确安装 JMF，并重新启动计算机之后，才能正常编译上述程序。

图 15.23 是用该程序播放一段 MPEG 文件时的运行截图。



图 15.23 程序运行截图

从程序 15.31 中可以看出，程序员只需要编写少量代码，就可以完成一个虽然简单但基本功能齐全，而且能够播放多种格式视频文件的媒体播放器，足见 JMF 的威力。

15.7 本章小结

本章全面介绍了多媒体数据处理的方方面面，包括声音、文字、图形、图像和视频。本章的多数程序都是直接利用了 Java 类库中的标准类来处理数据的，不需要自己对数据进行解码，所以程序都不算复杂。不过，尽管 Java 对处理多媒体数据提供了不少帮助，但如果真要编写出一个实用的、处理某一类多媒体数据的程序，仍然需要花费相当多的精力，而且需要阅读更为专业的书籍才能完成。